Tracking of a volleyball and 3d reconstruction of the trajectory

IACV Project 2020/2021

Image analysis and computer vision

Politecnico di Milano
Dennis Zanutto
marzo 2021

1. Description of the problem

The goal of the project is to detect and track a ball in a volleyball match, then trying to reconstruct the 3d trajectories would have ended the work.

Unfortunately, this task is quite tricky. As several papers on the internet say, see Diachen He & Li Li & Lina An (1), the main problems coming with this task are the following ones:

- The ball moves *quickly and irregularly*, making it difficult to predict.
- The small size of the volleyball in a match, makes the ball *lack of features points*. Moreover, it changes the size a lot during the game based on its position on the pitch.
- The texture of the ball continuously changes due to its *rotation*, making the problem of detecting it impossible.
- A very *noisy background* is always present. 12 players on the pitch, plus another dozen of people, like staff and referees, are caught by the camera. The audience, when present, makes the texture of the background even more complicate and changing over time. Even the big net, when it is moving makes the detection much harder.
- The 12 active players occupies most of the space on the image and for a lot of time the ball is *hidden* behind them.

To cope with these problems, common solution for professional purposes makes use of multiple high resolution camera, reconstructing a high fidelity 3d scene and then tracking the objects in the 3d world. For our project this is not possible. Due to covid measures, the sport activities at amateur level are suspended, thus it not possible to register on our own the matches.

Moreover, the video on the internet came with 3 big problems:

- *Low quality*, downloading from youtube limits the quality to 720x1280.

- *Low frame rate*, downloading from youtube makes the video compressed.
- *Single view* videos are present, there is no way to have more than one video for the same match.

For these reasons, the project focused on automatically tracking some serves from a specific video.

2. Standard solutions

Problems with standard algorithms

MATLAB computer vision toolbox offers several tools to track objects, but none of them suits well this problem.

I'll briefly explain why.

1. *Motion-Based Multi-Object Tracking (2)*:

This tool uses a foreground detector object. It uses a mixture of gaussian to learn the background and then computes a mask where objects in the foreground have the pixel set to 1, while the background is set to 0. This works well for steady and non noisy background. Moreover, the objects on the foreground should occupy a small part of the image and as already mentioned in chapter 1 this is not the case for volleyball.

After the segmentation between foreground and background, an analysis of the blobs present in the mask is done and the objects are assigned to tracks. Again, for this project the problem is that too many people are moving in a very small area, thus trajectories intersect continuously and the ball results as one of the less interesting object moving on the scene.

2. *People detector / Object detector:*

The idea behind the use of this tool was to train the weak learners to recognise players. Then, remove them from the foreground mask computed on the previous step, ideally leaving only the ball on the mask. The people detector (4) is a well trained system objects that works very well for static scenes and standing people, but for players it fails a lot. In the same frame the posture of the players varies a lot based on their position, thus it not possible to detect with high accuracy people.

The object detector that I trained over the video worked much better, but when removing objects from the foreground mask the ball disappears for almost every frame. This happens because usually the ball is inside the

box of a player and the detector has also some false positives, thus it removes more than it should.

3. *Histogram based tracker:*

This tools tracks object in videos that have a very characteristic color using HSV space. Despite the ball being yellow and blue, the above mentioned reasons made this tool useless. Primarily video quality and a non fixed texture of the ball.

3. My ad hoc solution

-Calibration

The first task to perform is the calibration of the camera. This operation is needed for a reconstruction in the 3d world and to understand the position of the camera with respect to the pitch. Moreover, no information about the camera is provided since the video is downloaded from the internet.

Volleyball courts have a standard regulation, where the coordinates of several feature points, even on different height planes, can easily be computed.

The pitch has 2 sides of 9x9 m. Furthermore, on the ground, it is always possible to see one line per side, called the attack line, that is 3 meters away from the center of the court.

The net has an height of 2.43 m for men's competition and of 2.24 for women's and junior's.

I decided to fixed the reference frame in the point called A, i.e., on the ground, on the center of the court. The x-axis will ride along the major length of the court, thus positive values of x will be on the right side, negative on the left one. Furthermore, for consistency, I decided to assign the number 0 to the right side and 1 for the other.

These informations give rise to some feature points with fixed coordinates:

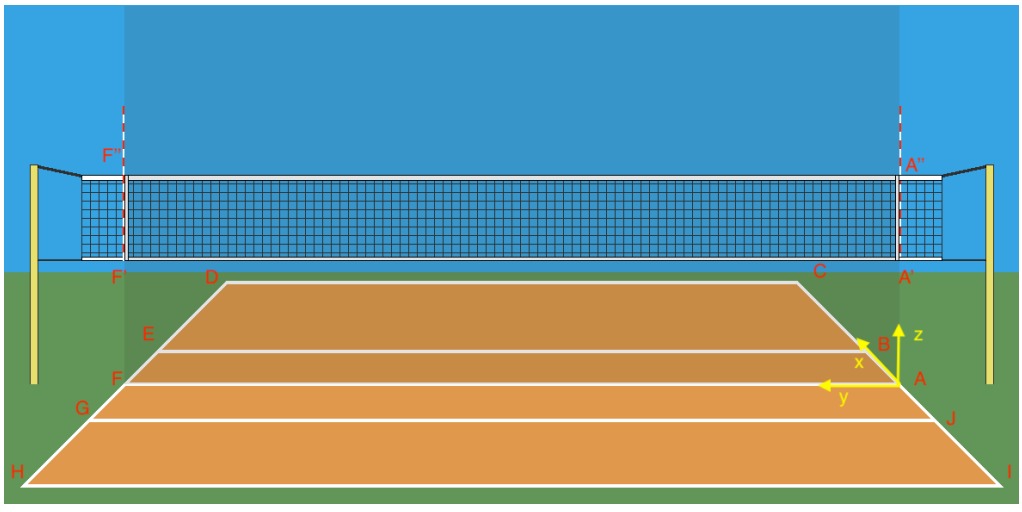

Fig. 1 - Key point definition

Then, some lines, from the projected pitch, have been selected using Canny and Hough. If, starting from this few lines, it is possible to find at least 4 of the aforementioned key points on the ground and 2 on another plane, then it is possible to apply a DLT algorithm that extract the Projection matrix from the real world to the projected image.

After the P matrix has been extracted it is immediately possible to extract the position of the camera with respect to the fixed reference frame. Moreover, by QR decomposition the 3x3 M matrix in the projection matrix can be decomposed in a triangular calibration matrix K and a rotation matrix R that explains the rotation of the camera.

This is the key part of the code:

```
% use DLT algorithm to get the projection matrix
P = estimateCameraMatrix( image_points, pitch_coordinates );
% the representation of points is inverted in MATLAB
P = P';

M = P(:, 1:3);
m = P(:, end);

% obtain rotation and calibration of camera
[R, K] = qr( M' );
% camera centre
O = -M\m;
```

And these are the extracted values:

$$R = \begin{bmatrix} -0.8710 & -0.1242 & -0.4753 \\ 0.4793 & -0.0034 & -0.8776 \\ 0.1074 & -0.9922 & 0.0625 \end{bmatrix} \quad O = \begin{bmatrix} -13.26 \\ -2.67 \\ 5.05 \end{bmatrix}$$

Back-projecting the key points we get an average error of 1.0455 pixels.

Fig. 2 - Key point back-projection

-Foreground detector

It is important to spend few words on this system object before explaining the algorithms, as it is key for most of the steps that will follow.

The foreground detector object(3) is a system object provided by the MATLAB computer vision toolbox. This tool uses a mixture of gaussian to learn the color of the background, then, calling its main method on a given frame will return a mask, where the pixels will have value 1 when the frame is different from the background and 0 when it shows the background.

The problem with the application of this technique in this project is the difficult texture of the background. The lines of the court and especially the net are above some steady color regions and they became hard to learn for this object.

This difficulty has been handled by using this object only in the region where it is more clear and its behaviour more predictable, i.e., away from the net.

As it is possible to see in the following figure, the region around the net is quite difficult to use because at least 4-6 players are present and the net itself causes difficulties in the recognition of moving objects.

Fig. 3 - Foreground detector problem

-Beginning of the action

Every rally in a volleyball match starts when the referee whistles. Thus, by listening to the audio of the video it is possible to extract the moment where the serve starts. For this project, this step has been skipped and a manually inserted starting time has been fed to the program.

During a serve, one player usually runs along the major axis of the court and hits the ball during a jump. The ball gets hit in the middle of the jump, usually at its highest point.

-Jump serve detection

This action is quite unique in those frames and can be extracted using the following tools:

- Cascade Object Detector
- Kanade-Lucas-Tomasi feature points tracking algorithm
- Sequential RANSAC

A. Cascade Object Detector

The MATLAB Cascade Object Detector(5) is a system object that consists of stages, where each stage is an ensemble of weak learners. The weak learners are simple classifiers called *decision stumps*. This object works well for objects with a fixed aspect ratio, like standing people.

So, the first step has been the training of an ad hoc object, called *ball_foundler*, that isolates the standing player in the region where the serve happens. This part can be found in the ball_founlders_trainer script.

Using the videoLabeler program in MATLAB, the labels around several players have been placed. The script automatically feeds these informations and it trains a new Cascade Object detector.

B. Tracking algorithm

The point tracker object(6) tracks a set of points using the Kanade-Lucas-Tomasi (KLT), feature-tracking algorithm. This is used to track the feature points of the objects found by the Detector. Then, these tracked points are summarised as the centroid of the player. This point is usually at the level of the hip bone, thus, for amateur level player at 0.9 m on average.

C. Sequential Ransac

Once the trajectory of the centroid of the player has been extracted, this can be studied to extract the key points of the jump in the serve.

Every jump serve has 3 characteristic parts:

1. The player runs toward the line, thus the centroid of its feature points will be following a straight line, some of oscillations are due to the arms moving.
2. The player jumps, the trajectory is almost a parabola, thus we need to look for a conic in the projected image.
3. The player lands, it usually goes inside the court to continue the rally, thus another straight line can be extracted.

From the set of points, these 3 objects can be extracted: 2 lines at the extremity and 1 conic in the center. The two points of the jump are the 2 points where the conic intersects with the lines.

The algorithm.

- *Load of the object detector* ball_foundler
- *Run of the object detector on the region of interest*
- *Several results appear, some are false positive: size and area are the main discrimination properties at this step*
- *For each remaining detection:*
  - *The feature points are extracted using* detectMinEigenFeatures
  - *The points are tracked using* PointTracker *for 55 frames*
  - *The centroid of the points in each frame is computed*

- *Some detections are pruned based on how the centroid moves during this frames: how much it moves allows to remove standing players or false object detection; the raw direction allows to remove also other moving players, that are not on the serve*
- *At this step only one detection is remaining, with the history of the centroid of its feature points*
- *From this set of points (the centroids history), the highest point of the parabola is extracted (in this case just looking at the highest point works, otherwise a normalisation of the history in pre-processing can help)*
- *Given the highest point and the 2 points on its left and right side in the history, the first conic is extracted*
- *Recursively, the following points in the history, both on the left and the right, are evaluated*
    - *When close enough they are included in the set of the conic and the conic is updated*
    - *When not close enough they are rejected for the conic and kept for the lines on the extremity*
- *Once the points of the conic have been removed, the remaining two set of points are extracted and the 2 lines defining the 2 segments are computed*
- *The conic is intersected with each of the two lines*
- *From the two solutions of each line and the parabola, just the closest one to the segment is selected*
- *This 2 points are the jump and the landing of the player; they are converted in the 3d world using the hypothesis that they are at 0.9 meters of height*
- *The instant and the x-y coordinates where the ball is hit are computed as the mean between the information of this 2 points*

Here some images extracted during this steps of the procedure:

Fig. 4 - Detected and tracked instances

Fig. 5. Player centroid detection

Legend:

-centroid history

-points of the conic

-intersections

-jump and land

Fig. 6 - Focus on jump trajectory

-Detection of the ball

The extraction of the jump has been performed and some informations are retrieved from this step. The algorithm analyses the frame in the middle between the jump and the landing of the player serving the ball. The best working solution is running the foreground detector on this frame.

In this frame, the position of the player is already known, thus it looks on the mask computed from the foreground detector for some circular objects above this position.

Since the players are usually far away from the net, this procedure works regularly and the ball is found with high confidence.

The algorithm.

- *Load of the frame and player position*
- *Run of the foreground detector on the region of interest*
- *Circle search*
- *Selection of the best match*

Here, what the algorithm sees and how it chooses the correct ball:

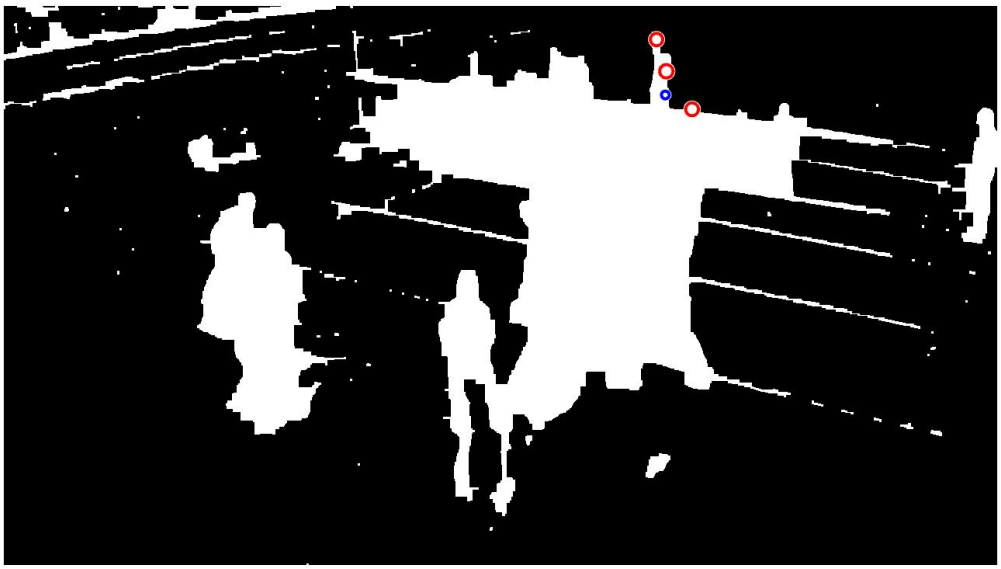

Fig. 7 - First ball position detection

-Tracking

The jump of the player has been tracked and the first position of the ball have already been acquired. After this two steps, the tracking of the ball starts.

For this task a class called history_tracker has been developed. This class contains some properties with important informations about the tracking of the entire action and several methods that takes care of assigning the correct matches of the ball.

Some of the properties are the following one:

- Coordinates, radius and bounding box of the ball in the image. Together with a label, that is defining the state of the detection, they store the history of the tracking.
- Total known and consecutive invisible are variables keeping track the success of the tracking process.
- Key points in the 3d world where the action takes place. This informations are needed for the following step in the 3d reconstruction.

To track the ball, the first step is the prediction of the ball, based on its history.

MATLAB offers a Kalman filter object that can do this task. Unfortunately, the ball moves very quickly, irregularly and on different vertical planes and this object it is not complex enough to solve these problems and tracking a ball moving along small and big conics. Thus, the first approach tried to simplify the prediction process, by predicting of the same step that the ball has just done.

This helped in the beginning, but the ball accelerates suddenly and in direction that cannot be predicted that easily due to projection theory. Then, to solve the problem, also the memory of the vector of instantaneous speed has been collected. In this way the prediction is performed using the last

position and the instantaneous speed. A more complex approach could have been, once the number of detections was high enough, that of computing an estimation of the trajectory and predicting the ball along it. This hasn't been implemented since the instantaneous speed method happens to handle well enough the conics.

Once the prediction has been performed we need to assign it to the best match that can be retrieved on the image.

It is important to mention that these analysis are performed only around the last position of the ball to save computational power. Some support functions like `extract_roi` helps in this task.

The algorithm looks at the circles in the region, starting from two mask. The one computed from the foreground detector and another one, called `stepper_mask` in the code, that is merely the difference between two subsequent frames. The first mask works well to find the ball in the first instants, but the second one is much better when the ball starts to move very quickly and around the net, so the cooperation of this two method is very important.

To maximise the chances, these two method runs in parallel. The script retrieves the circles in the foreground mask, the circles in the stepper mask and also compute another mask following the HSV color space of the ball, i.e., the yellow color.

First, it merges the results, the circles that are present in both the mask are the most likely to be the ball. Then, it neglects the fairest circles to prediction and keeps just the 10 closest ones.

After the pruning of the less likely results, it computes a cost function based on 3 parameters:

- distance from the prediction

- multiple detection (i.e. the circle is present both on foreground and in the stepper mask)
- quantity of yellow pixels in the region

The circle with the lowest cost function is the best match and it gets assigned to the history. Anyway, if the best match is not low enough, then no detection is assigned and the search is deepen using another technique.

If the circle analysis hasn't produced any good result, then it is very likely that the ball is moving so fast that its shape isn't a circle anymore. A workaround is that of looking for blobs in the stepper mask. If a blob with a similar area, in a position closed to the prediction, appears in the mask, it may be the deformed ball. Again, a similar algorithm using a cost function is performed. If a good enough result is found then it is assigned, otherwise the position of the ball remains under the label "unknown".

When the label is unknown it means that the ball hasn't been found, this can be due to many reasons, but usually it is because it is hidden behind some players. Anyway, the algorithm keeps track of how many times it fails to generate a match. If it just skips one or two frames, it continues and it recovers the position in the following frame, if it cannot, then it needs to take different actions.

The algorithm.

- *Load of the previous position of the ball and acquire the new frame*
- *Prediction of the ball*
- *Circle analysis and report generation:*
  - *Foreground analysis*
  - *Step analysis*
  - *HSV analysis*
  - *Merge of the circles*
  - *Selection of the best matches*
  - *Computation of the cost function*
- *If there is not a match*

- *Blob analysis and report generation:*
  - *Step analysis*
  - *Computation of the cost function*

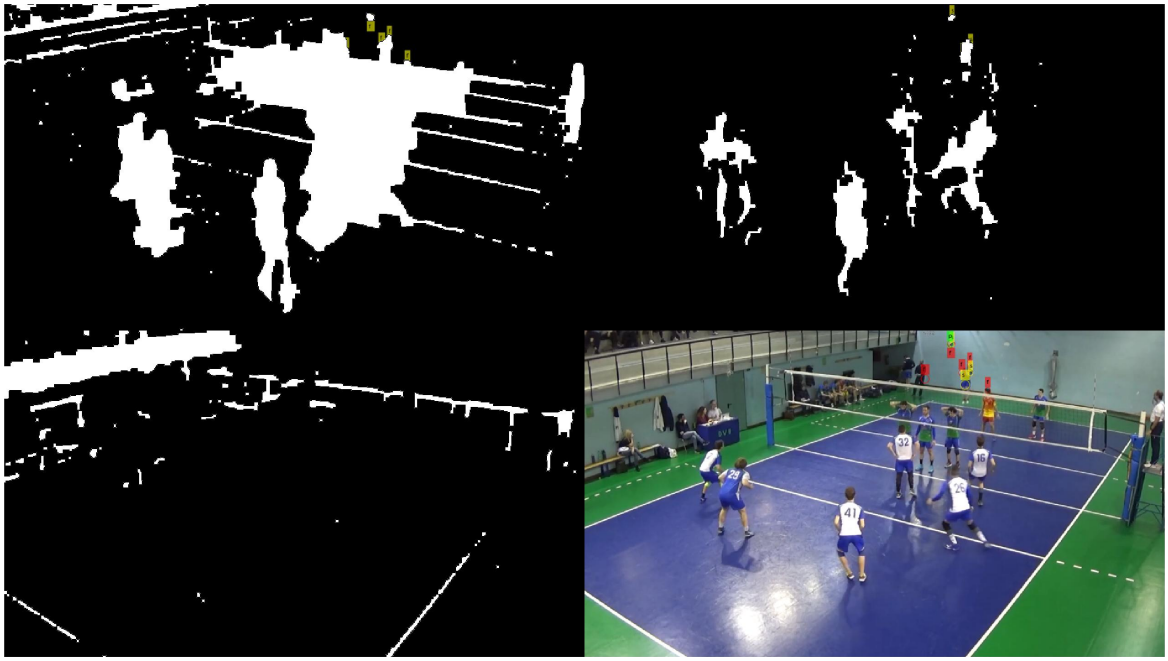

Fig. 8 - What the algorithm sees:

-top left: circles found from the foreground detector
-top right : circles from the stepper mask
-down left: HSV space
-down right: merge of the results and prediction

Fig. 9 - Focus on the possible matches: prediction, foreground, stepper

-End of the action

In a volleyball match a serve can end in 4 different ways:

1. Incomplete and the ball hits the net and doesn't go on the other side.
2. Incomplete and the ball goes outside the court.
3. Complete and the ball touches the ground.
4. Complete and the ball is received from a player.

The project is focused just on the serve, so in all of this 4 situations the tracking has to end.

When the ball hits the net, it is possible to detect automatically the situation because the ball changes suddenly its direction, it starts to fall vertically and it is in the region defined by the net for several detections. In this case, one constraint automatically rises, i.e., the ball is on the same plane of the net, thus, its x coordinate in the 3d world is 0.

Instead, the situation of the ball going directly outside the court or touching the ground it is more difficult to detect because there are no absolute key features. Even if it is possible to detect this situations happening, it is not possible to transform it in 3d because one constraint for the equation is missing. It is necessary to have at least one information about this. This two types of serving are avoided in the analyses.

The situation number 4 is possible to detect, but with less confidence. When the ball is hit by a player in the reception phase, their images in the mask merge and the ball is not detectable anymore for 4 (or more) frames in the video. Thus, it is possible to understand that the ball has been hit, because it is lost from the tracking process for a couple of consecutive steps, 5 in the case of the algorithm.

When this happens, an analysis of the last frames can detect the player.

In the algorithm, the object `video_handler` stores the `video_reader` and `video_player`, but it also keeps tracks of the analysed frames and of the information found on them.

By looking at the Harris features in all of this frames, it is possible to remove the static ones of still objects (remove_static_harrisfeatures) and keep only the dynamic ones, i.e. of moving objects. From an analysis of the masks, instead, the big objects that are detected are the players moving. Merging together this 2 sets of information, the algorithms detects the players and founds their centroids. The player closest to the trajectory and on the correct side of the court is assigned as the player receiving the ball. In this way we have the second reference for the 3d reconstruction, the position of the player receiving the ball.

The algorithm.

- *After tracking of the frame ends*
- *Check if one of the stopping conditions is met:*
    - *Ball hits the net*
        - *Retrieve the 3d point by using x = 0 as a constraint*
    - *Ball is lost*
        - *Start looking for the player*
        - *Get the last 10 frames and reports*
        - *Get the Harris Features from them*
        - *Remove static Harris features*
        - *Find possible players looking at the biggest blobs in the correct side of the court*
        - *Compute the centroid of the possible players*
        - *Find the closest player to trajectory*
        - *Retrieve the 3d point by using z = 0.7 as a constraint*
    - *Time expires*
        - *There is nothing that is possible to do*

Fig. 10 - Receiving player detection:

-green points: moving Harris features in the last 10 frames
-yellow boxes : big object detected moving
-red points: centroids of the green points of each box
-blue line: 2d fitted trajectory of the ball
-black point: selected centroid of the receiving player

-3d reconstruction

After tracking ends, the algorithm has the following informations:

- two 3d points of the plane where the trajectory takes place
- the history of the tracking in the image

From the two 3d points, the hypothesis that the plane where the action takes place is vertical, it is possible to extract the plane.

From the history of the tracking, it is possible to extract a conic describing the trajectory.

Together with the projection matrix, the cone containing the trajectory in the 3d world can be extracted.

By intersecting the cone and the vertical plane the trajectory in the 3d world is found.

The algorithm.

- *Find the plane π from the two points and additional constraint*

- *Find the conic C describing the trajectory in the image*
- *Find the quadric Q = P'*C*P describing the cone*
- *Find the 3d trajectory intersecting π and Q (it is done numerically)*

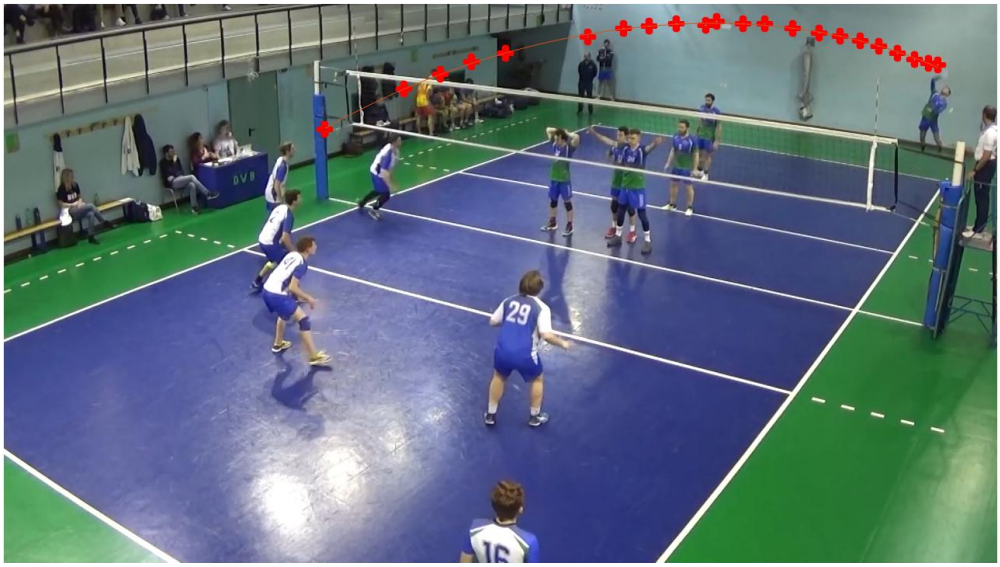

Fig. 11 - Fitted conic in 2d

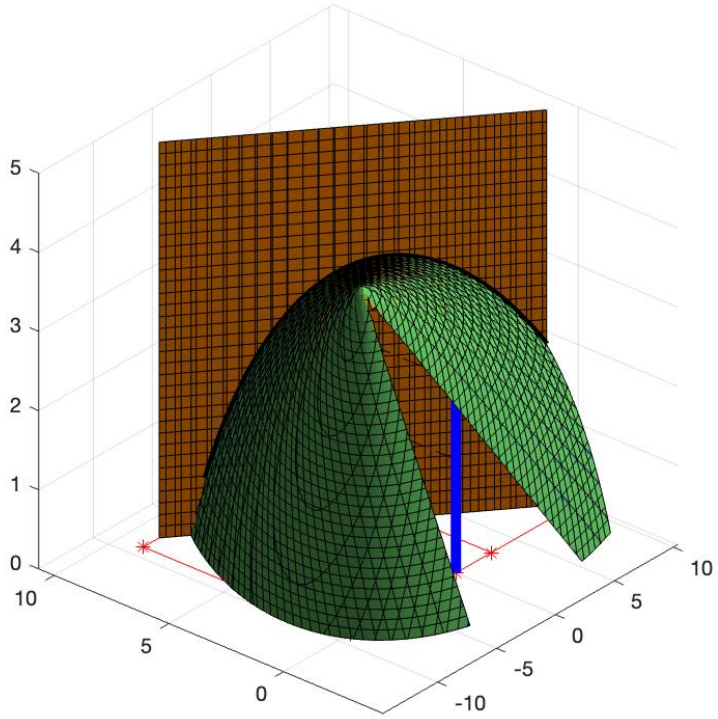

Fig. 12 - 3d cone Q and plane π

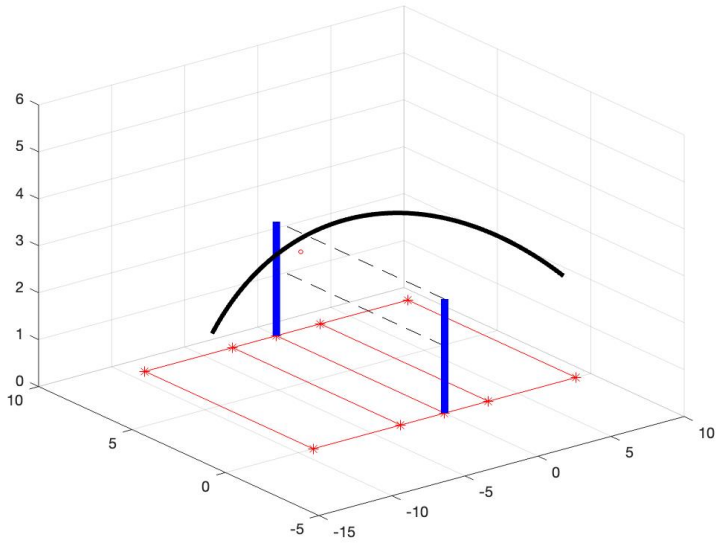

Fig. 13a - 3d trajectory

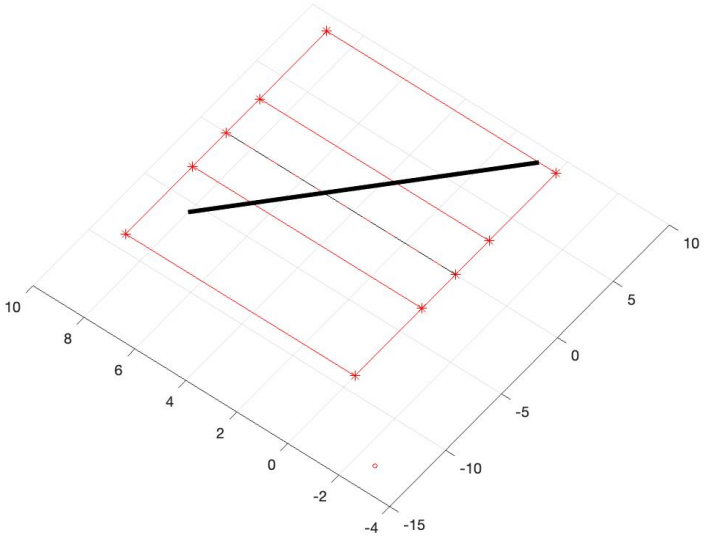

Fig. 13b - 3d trajectory

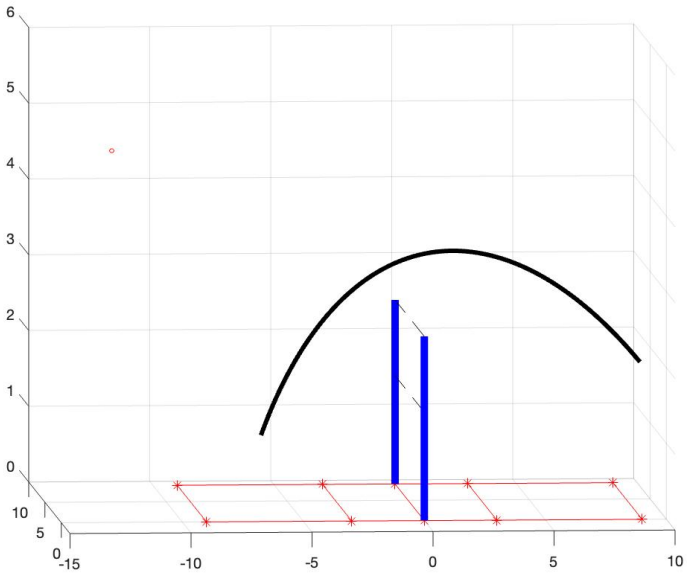

Fig. 13c - 3d trajectory

4. Conclusions and further development

The project has focused on only 3 actions and specifically on the serves of those.

Both the situation where the ball hits the nets or it is received by a player are correctly recognise.

A different approach could have been that of using more professional videos to improve video quality and recognise more easily the bar when it is in the air. On the other hand, this video has 3 more difficulties.

1. The camera rotates around its axis to follow the movement between the two sides.
2. The camera is usually set in a way that it is not possible to extract all the vanishing points.
3. Players hit the ball so hard that it becomes really difficult to track.

Both the first 2 problems can be solved, the first one by computing the rotation of the camera and applying this homography and the second one by knowing more informations about the camera and the setup. While the third one requires also professional cameras like tv one.

I preferred to focus on this video because the camera is in the best position to see the whole court and move to the 3d world.

A nice example of an alternative approach that just tracks the ball can be found at (7), although is written in python and it makes use of a tensor flow and neural networks.

A second version of the code could have the following feature: the program tracks both players and the ball, then at runtime it checks if a collision between a players and the ball occurs. When this happens this version of the program stops, while in a more advanced version it could just save the information, reset the kalman filter to track the new parabola and proceed.

One problem remains and it is the detection of the ball touching the ground or flying out of the court. These situation are hard to detect since they have no characteristic feature on the 2d projection. Using more camera and the 3d reconstruction of the scene would improve the results a lot.

5. How to run the code

The code is composed of several class and function and they need to be all in the same folder or added to the MATLAB Path.

It is sufficient to run the main, the file is called ball_foundlers_main.

It is necessary to have in the same folder the object actions.mat, that is the class that stores all the informations needed to detect the actions: starting side and starting time, name of the video and informations about the camera.

Before running the main it may be necessary to run the following lines to update the path (complete path plus name and extension) where the video is stored(if it is not on the same folder of the main):

```
a_h = actions_handler( cd,'actions.mat' );
a_h = a_h.set_videoname( path_of_the_video );
```

The first thing that the program asks is if the calibration is needed, press 0 to avoid this step. Useful informations are already stored in actions.mat and will be overwritten. Otherwise press 1 and follow carefully the procedure.

The program will learn the background.

The program will ask the user which action he/she wants to see.

The default available actions are 3, inserting a number between 1 and 3 will do the job. The 3d trajectory will be printed after the tracking. MATLAB version 2020a or following are suggested so that plotCamera works (Introduced in 2020a) .

If the users wants to add a new action:

```
a_h = a_h.add_action( );
```

Then insert starting side, starting time and maximum time for the tracking to end.

6. References

(1) D He & Li Li & L An, Study on Sports Volleyball Tracking Technology Based on Image Processing and 3D Space Matching

(2) Multi-Object Motion Based Tracking

(3) Foreground detector https://it.mathworks.com/help/vision/ref/vision.foregrounddetector-system-object.html

(4) People detector https://it.mathworks.com/help/vision/ref/vision.peopledetector-system-object.html

(5) Cascade object detector https://it.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html

(6) Point tracker https://it.mathworks.com/help/vision/ref/vision.pointtracker-system-object.html

(7) Python tracking without 3d transformation https://towardsdatascience.com/ball-tracking-in-volleyball-with-opencv-and-tensorflow-3d6e857bd2e7